Devin Trejo

ECE 3522: Stochastic Processes in Signals and Systems

Department of Electrical and Computer Engineering, Temple University, Philadelphia, PA 19122

I. PROBLEM STATEMENT

The assignment we will explore today is to create a simulation that will allow us to apply a pattern recognition system based around maximum likelihood classification. The entire process is a simulation thus we will be required to first generate our own Gaussian distrusted data, then transform it using principle component analysis as we explored in CA12, and lastly classify it. In our assignment we will focus on analyzing five different cases. In each case we will only change our σ_2 parameter. The parameters we analyze our laid out below:

Class 1:

$$\mu_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} ; C_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Class 2:

$$\mu_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} ; C_2 = \begin{bmatrix} \sigma_2 & 0.5 \\ 0.5 & \sigma_2 \end{bmatrix} \sigma_2 \rightarrow 0.25, 0.5, 1.0, 2.0, 4.0$$

As can be seen our first class parameter do not change. It is a Gaussian distributed multivariate array with equal energy in all directions. It has a mean that centers around [1,1]. Our second class energy changes but the correlation between our multivariate array stays constant at 0.5 Our second class will be center around [-1, -1].

From classification we use maximum likelihood classification. Before performing the classification we transform each set of data using principal component analysis. The process will convert our data sets so our data is not independently correlated. Thus our covariance matrix will have an off diagonal of zeros. By performing the transformation we can perform a Euclidean distance calculation (instead of a Mahalanobis distance) to the means of our data sets. What we wish to see is how the energy of our data correlates to the classification success of our data.

II. APPROACH AND RESULTS

We begin our analysis by create multiple cases to compare. The results are laid out below. We plotted both the original un-transformed data (left) with the transformed scatter plot of our classes (right).

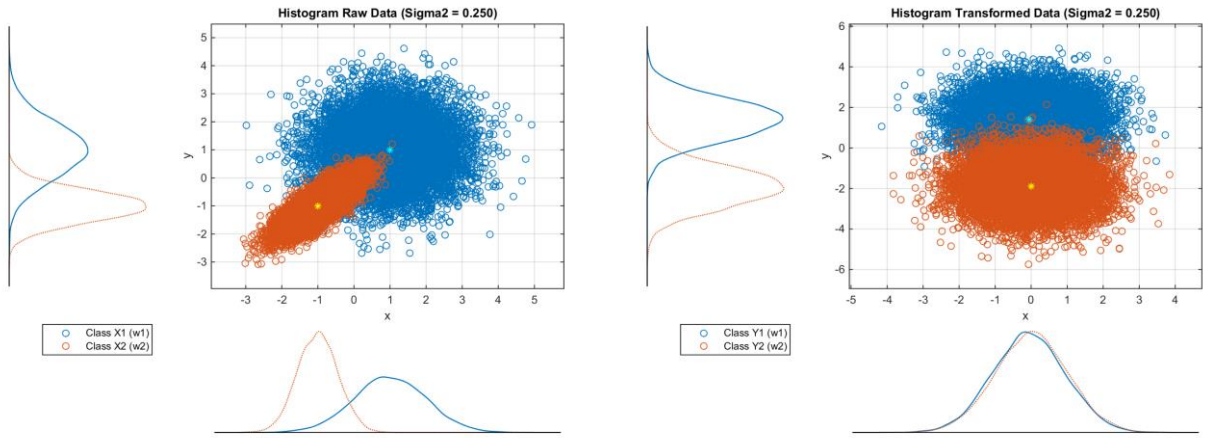

Figure 1: Scatter Plot Histogram for $\sigma_2 = 0.25$

Sigma2 = 0.250

Average Percent Diff transformed mean v actual: Y1 = 1.560, Y2 = 23.069

Y1: Class1 = 9520.000 Class2 = 480.000 || Y2: Class1 = 514.000 Class2 = 9486.000

Classification Error = 4.970

Figure 2: Mean Difference and Classification Results for $\sigma_2 = 0.25$

The above case is our first test. We see our two sets of data and how the principal component analysis transforms our data so that they become uncorrelated. Our transformation process is performed to the expected means of data sets where we see there is hardly any difference for our first set of data. For our second class our mean differs slightly. The error can be produced in our data creation process. We still see that even though we our mean is off by 23% our classification error rate is low for this case.

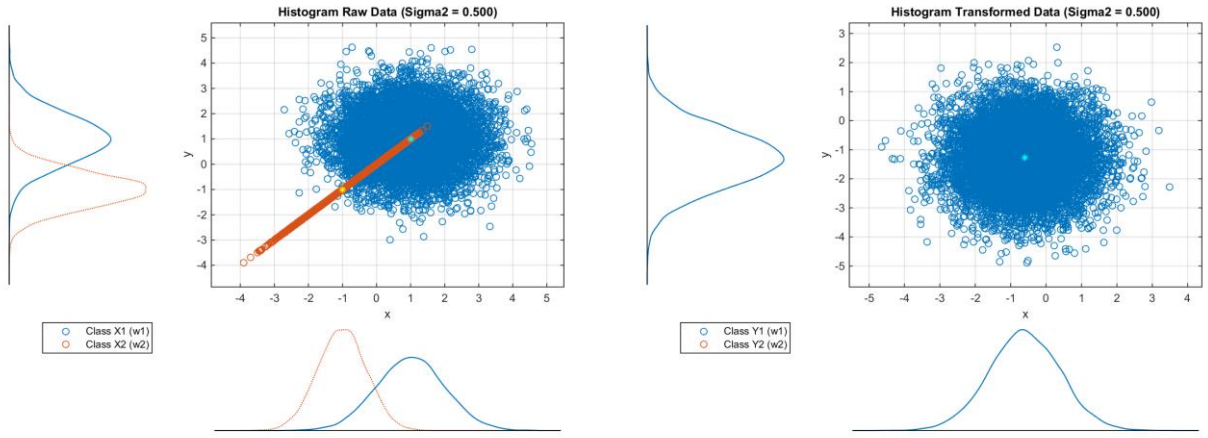

Figure 3: Scatter Plot Histogram for $\sigma_2 = 0.5$

Sigma2 = 0.500

Average Percent Diff transformed mean v actual: Y1 = 0.316, Y2 = NaN

Y1: Class1 = 0.000 Class2 = 10000.000 || Y2: Class1 = 0.000 Class2 = 10000.000

Classification Error = 50.000

Figure 4: Mean Difference and Classification Results for $\sigma_2 = 0.5$

The second case we encountered trouble in transforming that data. Since the data has equal variance and correlation in both directions our data is hard to un-correlation using principal component analysis. Thus we were not able to produce results for this case.

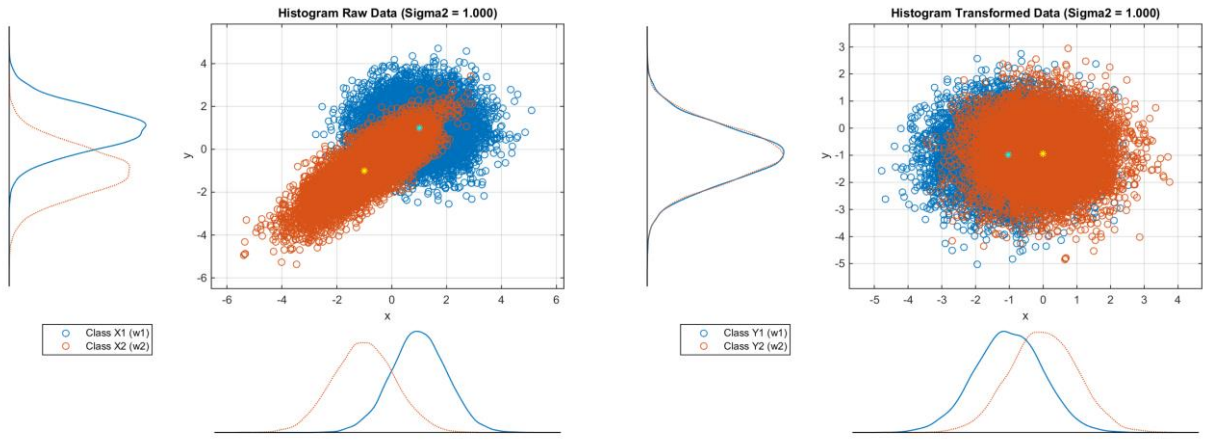

Figure 5: Scatter Plot Histogram for $\sigma_2 = 1.0$

Sigma2 = 1.000

Average Percent Diff transformed mean v actual: Y1 = 0.864, Y2 = 32.158

Y1: Class1 = 6892.000 Class2 = 3108.000 || Y2: Class1 = 3004.000 Class2 = 6996.000

Classification Error = 30.560

Figure 6: Mean Difference and Classification Results for $\sigma_2 = 1.0$

As we increase our energy for our second class so that it has equal energy as our first class we see it becomes harder to differentiate the two. In this case we produced a higher classification error rate. We observe how both data sets when transformed overlap heavily.

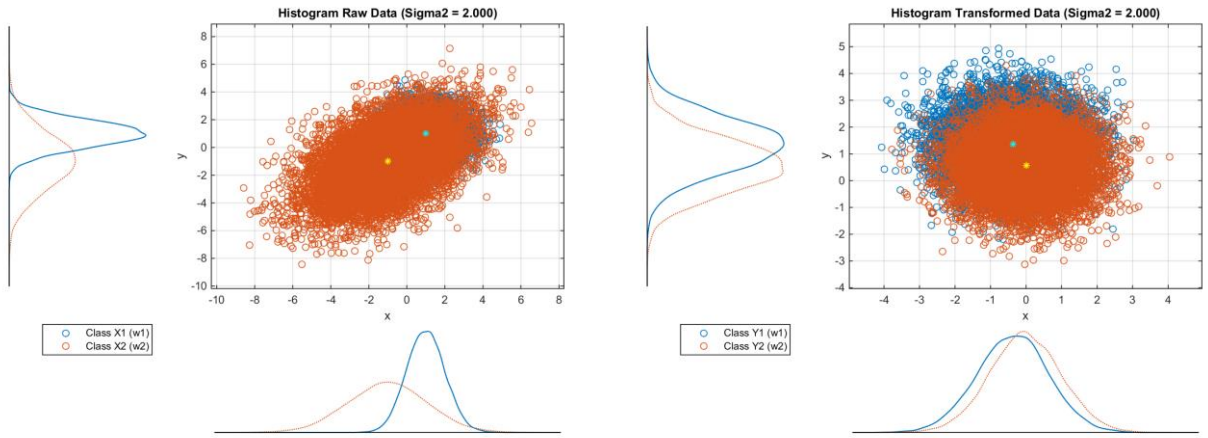

Figure 7: Scatter Plot Histogram for $\sigma_2 = 2.0$

Sigma2 = 2.000

Average Percent Diff transformed mean v actual: Y1 = 1.215, Y2 = 57.636

Y1: Class1 = 6684.000 Class2 = 3316.000 || Y2: Class1 = 3293.000 Class2 = 6707.000

Classification Error = 33.045

Figure 8: Mean Difference and Classification Results for $\sigma_2 = 2.0$

Similar to the previous case in this instance the two data sets overlap heavily thus we have a higher classification error rate. The maximum likelihood classification process still does have better than 50% success in differentiating the two data sets. For one our expected mean for our second transformed data set is off greatly from the true mean. Again depending on the number of data points our generator produces determines how actual mean differs from the true mean we asked it to generate.

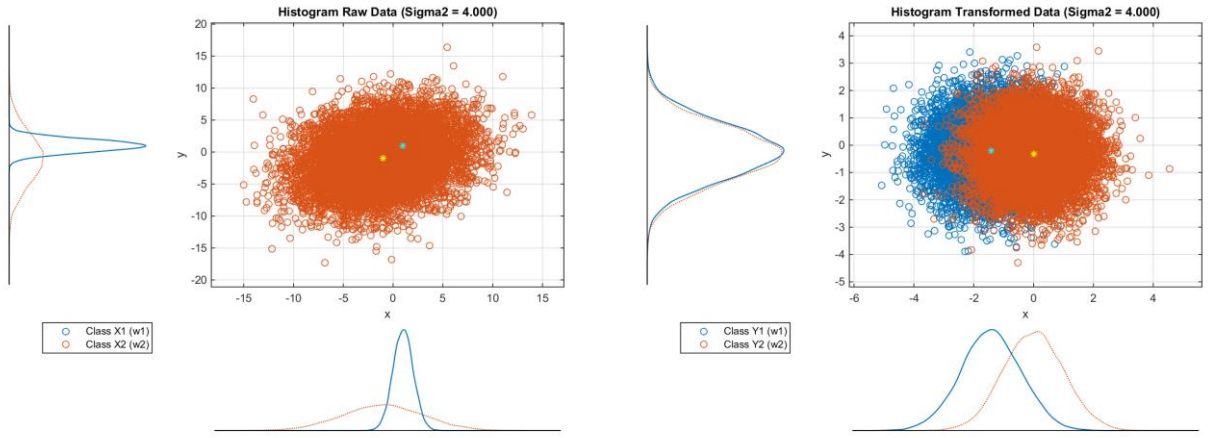

Figure 9: Scatter Plot Histogram for $\sigma_2 = 4.0$

Sigma2 = 4.000

Average Percent Diff transformed mean v actual: Y1 = 2.542, Y2 = 2.050

Y1: Class1 = 7633.000 Class2 = 2367.000 || Y2: Class1 = 2398.000 Class2 = 7602.000

Classification Error = 23.825

Figure 10: Mean Difference and Classification Results for $\sigma_2 = 4.0$

In our last case our generator produces data the centers around its true mean very nicely, thus our classification error decreases. We were successfully able to differentiate the two sets of data with an error rate of 23.85%.

III. MATLAB CODE

```
function out = get_grvs(mu, covariance, N)

    % Generate the random arrays
    out = randn(N,size(covariance,1))*covariance;

    % Add in Means
    % First make same size
    mu1 = ones(N,1)*mu(1);
    mu2 = ones(N,1)*mu(2);

    out = [out(:,1)+mu1 out(:,2)+mu2];
end
```

1 First function is ‘get_grvs’ which creates generates two normal multivariate distributions. We use the function to create our two classes. First we must create two arrays of normally distributed variables with size N. Next we multiply by the covariance matrix to fit it to the covariance matrix asked for by the assignment. Then we need to add the mean offset.

```
function h = plotHistScatter2(X1, X2, class1txt, class2txt)

    N = size(X1, 1);
    % To plot scatter we need to combine our two arrays
    X = cat(1, X1, X2);

    % Create a legend depicting the two arrays of X. Note this must be done
    % using cell arrays
    Xdescrip = cell(N,1);
    for i = 1:N
        Xdescrip(i) = {class1txt};
        Xdescrip(i+N) = {class2txt};
    end

    % Plot the combined scatter plot
    h = scatterhist(X(:,1), X(:,2),'Group',Xdescrip);
    grid on

end
```

2 We also created another function which allows us to plot histogram scatter plot called ‘plotHistScatter2’. We named it 2 since MatLab has a built in ‘scatterhist’ function but it only plots one set of data. To plot two sets of data we combine our matrices into one and create a cell array which will differentiate the two sets of data. If we pass the cell array to ‘scatterhist’ we are able to plot the two classes with a very cool histogram on both the X and Y axes.

```
function diff = perDiff(val1, val2)

    diff = abs((val1-val2)/val2*100);
end
```

3 Simple function that find the difference between two numbers.

```
function stoCa13(sigma2, N)

% sigma2 = .1;
% N = 1E4;

% Given Classes Values
mu1 = [1; 1];
C1 = [1 0; 0 1];
mu2 = [-1; -1];
C2 = [sigma2 0.5; 0.5 sigma2];


% Generate the multi-classes
X1 = get_grvs(mu1, C1, N);
X2 = get_grvs(mu2, C2, N);

% Plot Scatter
figure();
plotHistScatter2(X1, X2, 'Class X1 (w1)', 'Class X2 (w2)');
title(sprintf('Histogram Raw Data (Sigma2 = %0.3f)',sigma2))

% Plot New Means of Data
hold on
plot(mu1(1), mu1(2), 'c*');
plot(mu2(1), mu2(2), 'y*');
hold off
```

4 The main function for the assignment starts by generating our two classes. We take in two variables sigma2 and N which we will use in another script to pass it various sigma2 values for comparison purposes. Once we have our data we plot the scatter histogram of our data with their respective means highlighted with an asterix.

```
% Find actual Covariance Matrices
cov1 = cov(X1);
cov2 = cov(X2);

% Principle Component Analysis
% Eigen Value Analysis Returns Eigen Values in Ascending Order
[eigvec1, eigval1] = eig(cov1);
[eigvec2, eigval2] = eig(cov2);

% Tranformed Data
V1 = (eigval1^-(1/2))*eigvec1';
V2 = (eigval2^-(1/2))*eigvec2';
Y1 = (V1*X1')';
Y2 = (V2*X2')';

% Check to ensure PCA worked
Zc1 = cov(Y1);
Zc2 = cov(Y2);

% Transform Mean
mu1prime = (V1*mu1);
mu2prime = (V2*mu2);
meanY1 = mean(Y1)';
meanY2 = mean(Y2)';

% Find difference between actual mean and transformed means
diffmean1 = mean([perDiff(mu1prime(1), meanY1(1)), perDiff(mu1prime(2), meanY1(2))]);
diffmean2 = mean([perDiff(mu2prime(1), meanY2(1)), perDiff(mu2prime(2), meanY2(2))]);
fprintf(sprintf('Average Percent Diff transformed mean v acutal: Y1 = %0.3f, Y2 =
%0.3f\n',...
    diffmean1, diffmean2));
```

5 Using principle component analysis we are able to transform our two classes. First we find our Eigen values using the built in MatLab function. This we find our transformation matrix V1 and V2 for each class respectively. We can then transform our data. We compare the transformed means with the true means to if our transformation correlates to the true data.

```
% Plot new Scatter for Transformed Data
figure()
plotHistScatter2(Y1, Y2, 'Class Y1 (w1)', 'Class Y2 (w2)');
title(sprintf('Histogram Transformed Data (Sigma2 = %0.3f)',sigma2))

% Plot New Means of Data
hold on
plot(mu1prime(1), mu1prime(2), 'c*');
plot(mu2prime(1), mu2prime(2), 'y*');
hold off
```

6 Plot new scatter plot for our transformed data.

```
% Find distances between points for Y1 & Y2
% Start Counts at Zero
sameDY1 = 0;
class1Y1 = 0;
class2Y1 = 0;
sameDY2 = 0;
class1Y2 = 0;
class2Y2 = 0;

% Find distances between points Y1
for i = 1:N
   u1D = sqrt((Y1(i, 1)-mu1prime(1))^2+(Y1(i, 2)-mu1prime(2))^2);
   u2D = sqrt((Y1(i, 1)-mu2prime(1))^2+(Y1(i, 2)-mu2prime(2))^2);

   if (u1D == u2D)
       sameDY1 = sameDY1+1;
   elseif (u1D < u2D)
       class1Y1 = class1Y1 + 1;
   else
       class2Y1 = class2Y1 + 1;
   end
end

% Find distances between points Y2
for i = 1:N
   u1D = sqrt((Y2(i, 1)-mu1prime(1))^2+(Y2(i, 2)-mu1prime(2))^2);
   u2D = sqrt((Y2(i, 1)-mu2prime(1))^2+(Y2(i, 2)-mu2prime(2))^2);

   if (u1D == u2D)
       sameDY2 = sameDY2+1;
   elseif (u1D < u2D)
       class1Y2 = class1Y2 + 1;
   else
       class2Y2 = class2Y2 + 1;
   end
end
```

7 We now will attempt to classify our data by using Euclidean distance. For each data point in both our transformed data Y1 and Y2 we find its distance from the two data means. Whichever mean our data point is closest to we classify that data point to that class. We keep track whether is classified correctly or not.

```
% Find Classification Error
classErr = (class2Y1 + class1Y2)/(2*N)*100;

fprintf(sprintf('Y1: Class1 = %0.3f Class2 = %0.3f || Y2: Class1 = %0.3f Class2 = %0.3f\n
Classficiation Error = %0.3f \n\n', ...
    class1Y1, class2Y1, class1Y2, class2Y2, classErr));

end
```

8 As we classified our data we kept track whether it classified correctly or not. We print this to the console so that the user knows our classification error.

```
%% Program script
clear; close all; clc;

% Assignment Constants
sigma2 = [0.25 0.5 1 2 4];
N = 1E4;

for s = sigma2
    fprintf(sprintf('Sigma2 = %0.3f\n',s));
    stoCa13(s, N);
end
```

9 We wrote another script that allows us to sweep across various sigma 2 values. Sigma2 corresponds to the main diagonal for our second class covariance matrix.

IV. CONCLUSIONS

From our experiment we were able to clearly observe the process of maximum likelihood classification. Depending on the data you are working with will determine how accurately you are able to classify. In our tests we saw how our results depended greatly on our data being close to the expected means of [1,1] and [-1,-1]. We can correlate the assignment to sending data digitally over a communication channel. Digital data has values of zero or one thus we expect our data on the receiving end of our network to receive one of two values. However we know that as we send pulses across copper wires, resistance of the wires will cause the data to lose energy so any logic ones that you send down your network may look like zeros. Also since no system is perfect a digital zero will have some noise so its true value may not be zero. On your receiver you then must perform a similar process to what we performed in this assignment. There need to be a threshold between what you accept as a zero or one. Every bit you receive would need to go through some sort of classification process in order to determine what its values should be. We showed that there is error in the classification process thus you must design your system to tolerate or bit check these errors.

In this assignment we focused on only two classes of data. If we increase the number of classes it should be obvious that the classification process become harder. First you have the problem in determining how many classes you should be expecting. It may not always be known that the data you're looking at is produced by four different sources. Depending on the overlap between the classes will determine whether your thresholds are set far enough apart to section off your classes successfully.